(2)

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California




# THESIS

**IMPLEMENTATION OF A HYPERTEXT HELP SYSTEM FOR GLAD, A GRAPHICS LANGUAGE FOR DATABASE**

by

Lon M. Yeary

June 1989

Thesis Advisor: C. Thomas Wu

Approved for public release; distribution unlimited

Unclassified

Security Classification of this page

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a Report Security Classification: Unclassified | 1b Restrictive Markings |
| 2a Security Classification Authority | 3 Distribution Availability of Report: Approved for public release; distribution is unlimited. |
| 2b Declassification/Downgrading Schedule | |
| 4 Performing Organization Report Number(s) | 5 Monitoring Organization Report Number(s) |
| 6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (If Applicable): 52 | 7a Name of Monitoring Organization: Naval Postgraduate School |
| 6c Address (city, state, and ZIP code): Monterey, CA 93943-5000 | 7b Address (city, state, and ZIP code): Monterey, CA 93943-5000 |
| 8a Name of Funding/Sponsoring Organization; 8b Office Symbol (If Applicable) | 9 Procurement Instrument Identification Number |
| 8c Address (city, state, and ZIP code) | 10 Source of Funding Numbers: Program Element Number; Project No; Task No; Work Unit Accession No |

11 Title (Include Security Classification)
IMPLEMENTATION OF A HYPERTEXT HELP SYSTEM FOR GLAD, A GRAPHICS LANGUAGE FOR DATABASE

12 Personal Author(s) Yeary, Lon M.

| 13a Type of Report | 13b Time Covered | 14 Date of Report (year, month,day) | 15 Page Count |
|---|---|---|---|
| Master's Thesis | From To | June 1989 | 63 |

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| 17 Cosati Codes | | |
|---|---|---|
| Field | Group | Subgroup |
| | | |
| | | |

18 Subject Terms (continue on reverse if necessary and identify by block number)
On-line Help System, Hypertext, ACTOR, Object-Oriented, Graphical User Interface, GLAD, Database, Theses, Data bases, Computer graphics. (SPW)

19 Abstract (continue on reverse if necessary and identify by block number)
This paper explores the design and implementation of a help system for a graphical user interface named GLAD (Graphics LAnguage for Database). It examines help system design alternatives. Emphasis is on the implementation of a hypertext help system for GLAD using the Windows utility GUIDANCE and the object-oriented programming language ACTOR. Discussion includes the advantages of hypertext for on-line help systems. Keywords:

| 20 Distribution/Availability of Abstract | 21 Abstract Security Classification |
|---|---|
| [X] unclassified/unlimited [ ] same as report [ ] DTIC users | Unclassified |
| 22a Name of Responsible Individual: Prof. C. Thomas Wu | 22b Telephone (Include Area code): (408) 646-3391; 22c Office Symbol: Code 52Wu |

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page
Unclassified

Approved for public release; distribution is unlimited.

Implementation of a Hypertext Help System
for GLAD, a Graphics LAnguage for Database

by

Lon M. Yeary
Captain, United States Marine Corps
B.S., United States Naval Academy

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1989

Author: Lon M. Yeary
Lon M. Yeary

Approved by:
C. Thomas Wu, Thesis Advisor

David K. Hsiao, Second Reader

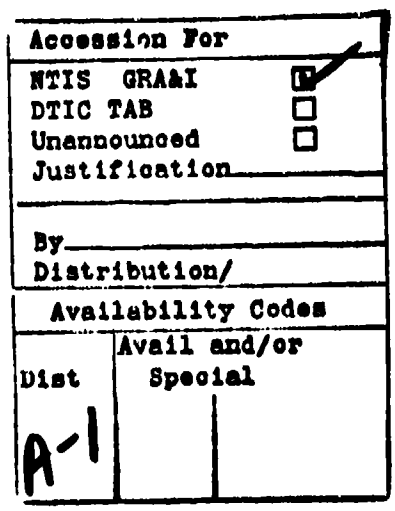
Robert B. McGhee, Chairman
Department of Computer Science

Kneale T. Marshall
Dean of Information and Policy Sciences

## ABSTRACT

This paper explores the design and implementation of a help system for a graphical user interface named GLAD (Graphics LAnguage for Database). It examines help system design alternatives. Emphasis is on the implementation of a hypertext help system for GLAD using the Windows utility GUIDANCE and the object-oriented programming language ACTOR. Discussion includes the advantages of hypertext for on-line help systems.





## TABLE OF CONTENTS

## LIST OF FIGURES

# I. INTRODUCTION

## A. OVERVIEW

GLAD (Graphics LAnguage for Database) is a coherent, graphics-oriented interface for data manipulation and program development with a database (WU, 1989). It was originally proposed in 1985 by Dr. C. Thomas Wu of the Naval Postgraduate School, Monterey, California. This unified interface is designed for easy learning and use while providing a comprehensive visual representation of a database schema. A major design principal of GLAD was that the user should have access to additional information whenever desired (WU, 1989). This thesis proposes a context-sensitive help system utilizing hypertext to augment the information available to GLAD users and increase the overall usability of the database interface.

## B. BACKGROUND

In order to manage the vast amounts of information generated in today's complex world, it is vital that a simple, yet powerful database management system be available. Desirable systems would not require a costly database specialist to program every modification. A second desirable characteristic of a database management system is that it must be easy-to-learn and use. GLAD research has been motivated by the desire to develop a theory identifying the best method of graphical user interface for

interaction with a database. It was the lack of an easy-to-learn, easy-to-use query facility for accessing databases that led to the proposal for GLAD. WU believed that by proposing and comparing as many graphical interfaces as possible, a "best" methodology could be established. GLAD is unique in that it is not based upon a specific data model, however, it provides a visual representation of the four most widely used abstraction concepts: aggregation, association, classification and generalization (WU, 1987, p. 3). Through the use of simple visual representation of a database schema, GLAD is easy to learn and use. This is critical for wide-spread use of a database management system. We believe this model for database management holds the best potential as an end-user interaction tool.

## C. DEVELOPMENT OF GLAD

There have been two cycles in the development of GLAD. Version .01 developed a rudimentary prototype to see whether a full implementation of GLAD using object-oriented programming and ACTOR[1] was feasible (WU, 1989). The second cycle of GLAD added a data definition component, bitmap display capability, and updated the facility of the first version. Version .02 has added some classes to increase the functionality of this and future versions. Originally, plans were made to develop GLAD as a system compatible with Sun Microsystems' workstation. A study showed this to be infeasible (Wu, 1989). Attempts were made to implement GLAD using C;

[1] ACTOR is a registered trademark of The Whitewater Group, Inc.

however, it was found that available routines were too low-level and very difficult to use. For example, the code required to display a single scroll bar required ten parameters (WU, 1989). Due to budget and manpower constraints this approach was deemed unwieldy. A rapid prototyping tool was critical to the success of this visual interface; therefore, a search began for a development tool which provided a quicker and easier method of developing windowing routines. This development tool needed to be capable of rapid prototyping and provide extensibility. Naval Data Automation Command, a GLAD sponsor, imposed the requirement that the system be capable of running on an IBM PC/AT compatible MS-DOS machine. It was also considered desirable to keep the system capable of being ported to other platforms such as Unix and Macintosh. It was felt that features of an object-oriented programming language such as inheritance, encapsulation and polymorphism would best facilitate the most expedient implementation of GLAD. After exploring Smalltalk and other similar languages, information on ACTOR was found. ACTOR appeared to meet most of the requirements specified above.

ACTOR is an object-oriented language introduced in 1987 for developing Microsoft Windows (MS-Windows)[2] applications. It is a language that appeals to intuition and does not depend upon the user possessing a wide background in traditional computer science topics (Rowell, 1988, p. 77). In 1988, a GLAD prototype was successfully

[2]MS, Microsoft and Windows are trademarks of Microsoft, Inc.

implemented using ACTOR (Williamson, 1988, p. 32). ACTOR was selected as the language to be utilized for the development of GLAD. Interactive in nature, ACTOR features several powerful tools that provide a sophisticated programming environment for developing MS-Windows applications (WU & HSIAO, 1989, p. 3).

Among the tools provided in the ACTOR programming environment are a Browser for viewing, modifying, and creating classes, an Inspector for viewing object structure, a Debugger for debugging ACTOR code, and an Editor for simple text editing. ACTOR allows static and dynamic binding of variables which are useful for rapid prototyping of application software while not compromising the efficiency of the final software product. ACTOR is designed to run within Microsoft Windows on an IBM compatible personal computer with at least 640K of memory and a hard disk (Duff, et al., 1989, p. 1). An EGA or VGA color monitor and extra memory are highly desirable, though optional. The prototype for GLAD was developed using ACTOR on an AT-compatible machine with an EGA color monitor and a 386 machine with a VGA color monitor.

GLAD provides an object-oriented data model which provides a close relationship between the structure of the database to be modeled and the logical entities used within the object-oriented programming environment. Wu and Hsiao (1989) have identified six design principles for the development of GLAD. The program should be able to:

- Provide more information when asked.
- Recover from the unintended or erroneous operation.
- Perform the same operation in more than one way.

- Perform logically equivalent operations in a consistent manner.
- Display multiple information at the same time.
- Display multiple views of the same information.

## D. MOTIVATION FOR DEVELOPMENT OF A HELP SYSTEM

While it is theoretically possible to develop a computer program that is so well designed that no user assistance is ever required, practically speaking, it is impossible. The philosophy of the GLAD project is that the end-user should have easy access to any information that will permit the maximum possible usability. Design principle one states that the user should have access to additional information when requested. It is envisioned that a user will be able to obtain assistance from within the GLAD environment concerning any operation, term, or process. The purpose of this thesis is to research, design and implement a help system that meets the design principles proposed by Wu which will promote even greater end-user usability for GLAD.

## E. ORGANIZATION

The remainder of this thesis is organized as follows. Chapter II discusses help system design alternatives. Chapter III discusses the design decisions made in the development of the current GLAD help system. It includes a discussion of constraints imposed by the GLAD environment and the decision to incorporate hypertext into GLAD. Chapter IV discusses the implementation details of the GLAD help system. It includes a discussion of the utilization of dynamic link libraries with ACTOR and

the incorporation of the hypertext system GUIDANCE[1]. Cor lusions and future research for GLAD are discussed in Chapter V.

[1]Guidance is a registered trademark of Owl International, Inc.

# II. HELP SYSTEM ALTERNATIVES

## A. WHAT IS A HELP SYSTEM?

A help system is a program (or several programs) that assist the user in the operation of another, usually, larger program. The help system can either be a separate program that runs concurrently with the larger program or, as is more common, a program that is completely integrated within the larger application (Kearsley, 1988, p. 3). Help encompasses a continuum of assistance to the user, from a memory jogger of available commands to on-line documentation which includes or supplements the user manual. Help includes error messages, tutorials to assist in the learning process, and detailed explanations of terminology. Help also includes prompts which guide the user through the available choices within the program.

Help for a computer user comes in two basic forms. It can be printed (hard-copy) which includes user manuals and reference guides, or it can be on-line. Tutorials are another form of computer help; they can be printed, on-line or both. There are advantages and disadvantages to both printed and on-line help. Research has shown that speed and comprehension is greater using printed materials than when reading from a CRT (Shneiderman, 1987, p. 360). Printed help allows the user to refer to his work and the help information simultaneously without interrupting the program.

On-line help provided in a windowed environment will also provide this capability. On-line help has the advantage of being available any time the program is running. While at the terminal, the user does not have to be concerned with locating the manual, and manifold users within an organization will not have to purchase multiple copies of documentation. Additionally, on-line help is generally current. Unlike a user manual which may be outdated with respect to the software, on-line help is usually updated with the software; therefore, it is more likely to be current.

Regardless of the advantages of on-line help systems, it is clear that they cannot completely replace the printed user manual. A "perfect" help system would be of no use to a naive user who did not know how to turn the computer on. Instead, an on-line help system can be thought of as a method of enhancing or augmenting the information available to a program user. This thesis addresses the development of an on-line help system. It will not discuss the need to develop a good user manual for GLAD.

Dumas (1988, p. 50) proposes seven principles for a good interface design. One of these design principles is the incorporation of an on-line help facility. A well designed help system contributes to the overall usability of a program by reducing the time it takes to complete an on-line function and by reducing the number of errors made. Jackson and Lefrere (1986), Shneiderman (1987), and Kearsley (1988) all discuss research which confirms the advantages of a well designed on-line help system. Improving the design of the help system will improve its benefit to the user.

Improved user performance and user satisfaction can result from having a skillfully planned help system integrated into a software program. If only by providing increased confidence for the user, a help system can be beneficial, speeding the process of familiarization (Jackson and Lafrere, 1986, p. 64).

To be truly useful, an on-line help system must be easier and quicker to use than a manual (Roberts, 1970, p. 547). In addition, it must be accurate or it will destroy any confidence the user might have in it (Killory, 1987, p. 19). On-line help is only useful if it is capable of providing assistance to a user to derive or debug a plan of action (Jackson and Lefrere, 1986, p. 63). When users request help, normally they are seeking a specific explanation for an immediate problem. A help system cannot assume that a user possesses a specific proficiency level. The help system must present information commensurate with the user's level of expertise.

## B. TYPES OF HELP SYSTEMS

### 1. Static versus Dynamic

There are many design alternatives available when developing an on-line help system. One of the first considerations is whether the help offered should be static, that is independent of where the user is in the program and any previous actions, or dynamic, that is dependent on where the user is and what the user's previous actions were. Static help could be thought of as a kind of "on-line glossary of terms" (Kearsley, 1988, p. 14). A list of terms is provided with a brief explanation when a

user requests help. Regardless of the function being performed during program execution, the same information will appear whenever help is requested. WordPerfect's[4] F3 key is an example of a static help system. When F3 is pressed, the user is presented with the same list of command options regardless of where he is in the program. Static help allows requests for specific terms, however, the user must know what term requires further explanation. While it is sometimes helpful to provide every detail to the user searching for a solution to a problem, it is possible to provide so much information that the user becomes lost or distracted. If the search takes the user through unnecessary information, the user may be delayed in actual task completion.

Dynamic help that is sensitive to the sequence of user requests or actions can be developed. Dynamic or context-sensitive help offers assistance based on the function being performed during program execution. For instance, if the user is in the edit mode of a program and requests help, an explanation of edit commands available user would be provided without listing unrelated commands. Dynamic help can be as simple as an explanation of an error message or it can be more sophisticated, providing detailed information of program options based on where the user is in the program or the function being performed. By increasing the degree of context sensitivity, the usefulness of the help is increased.

---

[4]WordPerfect is a registered trademark of WordPerfect Corporation.

A powerful form of dynamic help is a dialog between the user and the system. The dialog guides the user through the problem providing step-by-step instructions (Kearsley, 1988, p. 16). Dialog help is difficult to develop. It requires the incorporation of assistance for every possible user action or question into the help system. These help systems require extensive error checking to ensure correct user input. SHERPA, a help system developed by ComTrain for LOTUS 1-2-3, is a good example of a dialog-type help system which uses prompts to assist the user through a series of functions (Kearsley, 1988, p. 35).

### 2. Multi-level Help System

It is clear that providing a complex step-by-step procedure for every user may not be desirable and may even slow processing time considerably for the experienced user. Therefore, another design alternative is to provide multiple levels of help, giving more detailed information for each successive level of help requested. In addition to increasingly detailed information, successive levels of help would provide examples, qualifiers on use, or descriptions of related commands. Some programs such as MicroPro's WordStar[3] allow users to set the level of help when the program is initiated. WordStar users have the choice of four levels of help, ranging from a full menu of choices on screen at all times to no help prompts displayed. Another example of a multi-level help system is one that utilizes different terms to request different

[3]WordStar is a registered trademark of MicroPro International Corporation.

levels of help, for example, "Define," "Explain," "Example," or "Limitations" (Kearsley, 1988, p. 17). Having different levels of help available is an attempt to meet the specific needs of various users including the novice, the occasional user needing only a memory aid, as well as the experienced programmer.

### 3. System-Initiated Help

Help requests may be initiated by the user or by the system. System-initiated help, usually given as advice or as a suggestion, is frequently triggered by an error condition. For example, if a user types a command that is inappropriate for the current function, the system would provide a list of all the correct alternative commands. Some users perceive system-initiated help as an interruption or as a delay in using the program, and may even consider the messages as "nagging". System-initiated help is necessary in some systems to prevent users from making "fatal errors", especially in systems where errors could result in catastrophic consequences. System-initiated help also assists the user by pointing out shortcuts that may have previously been unknown. This could prove useful to the novice and experienced user alike. For example, the system may be designed to inform the user when he is using several operations to accomplish a task which should be incorporated into a single operation.

If system-initiated help is utilized, it is recommended that the user be given the option of turning it off or specifying the level of message desired (Kearsley, 1988, p. 20). Specific messages of highest priority could be left on, while allowing the user

to turn off advisory messages that may not be pertinent to their use. In order to provide the advantages of system and user-initiated help, a system can be designed which incorporates both.

### 4. User-Initiated Help

User-initiated help can be activated through a variety of means, including typing a help command, pressing a designated help key, or by selecting a help option from a menu. The advantage of this method is that the help may be obtained independently of typed input. The disadvantage is that the user may not remember which key is the help key unless a method for keeping a reminder on screen is utilized. Typing a help command may be desirable in that it allows for complex syntax to be used to take the user to a specific area of needed assistance; however, the user will have to remember the correct syntax of the help command.

Utilizing menus with a help option has several advantages. The user will have command options displayed in the menu eliminating the need to remember the command choices. In addition, the user is reminded every time he looks at the screen that the help option is available. Menus have the same problem as designated help keys in that both require the user to specify the topic on which help is required. Menu help systems allow the user to select from an index of topics. The need for greater specificity is eliminated if the system is context-sensitive. The fact that a system is menu-driven does not necessarily imply that it is effective. "In fact, poorly

designed menus are one of the most common problems with application software" (Dumas, 1988, p. 60).

Menu-driven systems are becoming increasingly popular. Microsoft's Word[6], MicroPro's WordStar, and Wordperfect Corporation's Wordperfect have all released menu driven systems. Aids in converting existing programs to menu-driven programs are already on the market. Macintosh's iconic menu system is very popular due to its ease of learning. While this is primarily a movement toward improving user-interface design, it directly affects help systems available to the user. Commands will be on screen, eliminating the need to remember the exact syntax of the command or the proper command key.

### 5. Screen Options

Formatting of the screen, to a large extent, will be dependent upon the capabilities of the operating system being used and the constraints of the programming environment. It must be determined if the help message will be located within the existing screen or if a separate screen will appear. Critics of on-line help systems state that it is difficult for users to remember what the help screen states when they have to switch back to the operating screen (Weiss, 1985). Help messages that are displayed on part of the operating screen, for example at the bottom or on the next available line, may be limited in the depth the help message can display due to limited

[6]Word is a registered trademark of Microsoft, Inc.

space. It also may be desirable, as mentioned earlier, to keep on the screen the information needed to access the help system, i.e., "**F-1 = Help**". This is usually located either at the top or the bottom of the screen.

Operating systems that support bitmapped graphics allow for independent windows that can eliminate many of the problems in screen formatting. Windowing capabilities allow for both the operational screen and the help screen to be displayed simultaneously. The operating system being utilized will dictate if the windows will overlap each other, potentially blocking part of the other screen, or if the window can be moved and adjusted in size to allow the user to have access to both sets of information. Some of the problems that have been identified in the use of windows are that excessive window manipulation may be time consuming and distracting, the need for window borders consumes valuable screen space, and small windows with large amounts of information may lead to excessive, bothersome scrolling (Galitz, 1989, p. 99).

Another consideration in screen formatting is whether the help screen should page or scroll through the information if it exceeds the space available. Both methods should keep in mind that the user should have to exert minimal effort to go both forward and backward in retrieving information to minimize any delay.

### 6. Extensibility

Extensibility can be defined as "the ease with which software products may be adapted to changes of specifications" (Meyer, 1988, p. 5). Extensibility is very important to the design of a good help system. The information contained in the help must be modifiable. As the software program evolves, the help system will need to keep pace to remain useful. The user may also want to "personalize" the help system, adding more information to the help message to reflect the way the command is used (Kearsley, 1988, p. 23). A system of preventing inaccurate information from being added to the help system must be utilized in order to maintain the integrity of the help system. If a user altered an existing help message inaccurately, it would ultimately affect many users who also utilize the program. The proposed solution to this problem is to allow additions to the help system while preventing modifications to the original content of the help system.

## C. HYPERTEXT

Hypertext has been hailed as ideally suited to help systems (Kearsley, 1988, p. 19). While hypertext has been around for over two decades, it has recently enjoyed an upsurge in interest (Conklin, 1987, p. 32). With hypertext, windows on the screen are associated with objects in a database. Links are provided between these objects both graphically and in the database as pointers. Major terms or control options can be selected that will then be further explained. Any term or graphic display can be used

to provide further help. Hypertext is capable of performing "high-speed, branching transactions on textual chunks" (Conklin, 1987, p. 32). It cuts across traditional boundaries; at the same time it is a database method, a representation scheme, and an interface modality (Conklin, 1987, p. 33). It is a way for high speed, automated browsing through information (Guide, 1988, p. 3). Any text item can act as a "trigger" for various kinds of help information and actions (Kearsley, 1988, p. 43). Owl, International introduced "Guidance" in 1988. It is, on its simplest level, a context-sensitive on-line help system that uses hypertext. It is a "read-only windows application with a simple interface so that it can be driven by a host application" (Guidance, 1988, p. 5).

Although there are many advantages to using hypertext, Conklin (1987) lists these nine as significant:

- Ease of tracing references.
- Ease of creating new references.
- Information structuring.
- Global views.
- Customized documents.
- Modularity of information.
- Consistency of information.
- Task stacking.
- Collaboration.

He also identifies two major problems in the use of hypertext. First, a user can get lost in the document because there is not the traditional linear way of moving through the document. This "disorientation problem" can be corrected through technical solutions, such as the use of a graphical browser that can "map" the network that is

linked. Secondly, "cognitive overhead" or the additional effort needed to juggle several tasks at once may become troublesome with hypertext. However, utilizing a rapid cross-reference node can ease this problem, as can an instantaneous one to three line explanation in a pop up window as a side reference.

Through the use of hypertext, it is possible to incorporate many of the concepts in help design alternatives that promote maximum usability. Hypertext allows for both static and dynamic context-sensitive help on a multi-level basis. It can access specific information needs through the use of major terms which serve as pointers or triggers to elaborate further information.

## D. SUMMARY

Help system design provides several alternatives. Tradeoffs will be required between space, time and functionality. Design decisions needed in creating a help system have been delineated. Major decisions include:

- Utilizing static or dynamic help.
- Making the help system multi-leveled.
- Making help user or system-initiated.
- Selecting screen options.
- Incorporating extensibility.
- Utilizing hypertext.

Hypertext appears ideal for the development of a help system. A good help system will incorporate the best features of each design alternative.

# III. THE GLAD HELP SYSTEM

## A. DESIGN CONSIDERATIONS

The objective of this project is to develop a help system for GLAD. Inherent within this project are certain constraints. First, the program must be able to run on a IBM PC compatible computer as requested by the project sponsors. Second, GLAD is a Microsoft Windows[7]-based application. It is, therefore, logical to develop a help system that is Windows-based. Windows provides certain conventions and capabilities which influence the design of a help system. A few examples of these conventions and capabilities are the use of a mouse, adjustable windows, a common menu presentation and a common method of closing windows. GLAD is a graphical interface. In order to maintain the desired consistency, it is logical that the help system provide graphical capabilities as well. In addition, Windows easily facilitates the presentation of bitmapped graphics.

The design principles for Glad were established prior to the design of the help system. These six design principles must be adhered to in order to keep the help system consistent with GLAD. Previous developers of GLAD have suggested that it

[7]Microsoft Windows hereafter referred to as "Windows".

should include a context-sensitive help system (Williams, 1988, Wu and Hsiao, 1989). These suggestions have directed the design of the GLAD help system.

Another consideration in designing this help system is that GLAD is still under development. A quality method of implementing the help system was needed which would provide extensibility and easy modification. One final constraint which must be considered when working on any project of this nature is time. The time available for the completion of this project was finite.

## B. GLAD DESIGN CHOICES

Given the above constraints and the design principles for GLAD, certain choices for the help system were obvious. First, since GLAD is a Windows-based application, the help system for GLAD must also run in windows. Second, since previous developers of GLAD specified the need for context sensitivity, a dynamic help system capable of sensing the location of the user within the document is preferred. Third, to be useful to both the novice and the experienced user, the help system must be multi-leveled, allowing the user to view as much information as required. Naturally, the system has to run on the same hardware as GLAD.

The requirement of having the help system run within the Windows environment is, in many ways, a design advantage. The Microsoft Windows development environment eases design decisions through its conventions and capabilities. The first design principle of GLAD states "be able to provide more information when asked".

The Windows programming environment establishes a conventions which provides a convenient method of incorporating this principle into the help system. Programming Windows recommends the right justified position on the menu bar as the standard position for help (Petzold, 1988, p. 361). It also recommends enclosing the HELP menu selection within a box which separates it from other menu selections (Petzold, 1988, p.406). Windows provides simple methods which allow menus to be developed utilizing these conventions. F1=HELP appears in the standard position enclosed in a box on every menu bar of GLAD, see Figure 3.1. Help can be accessed by clicking the left mouse button on F1=HELP within the menu bar from any window of GLAD.

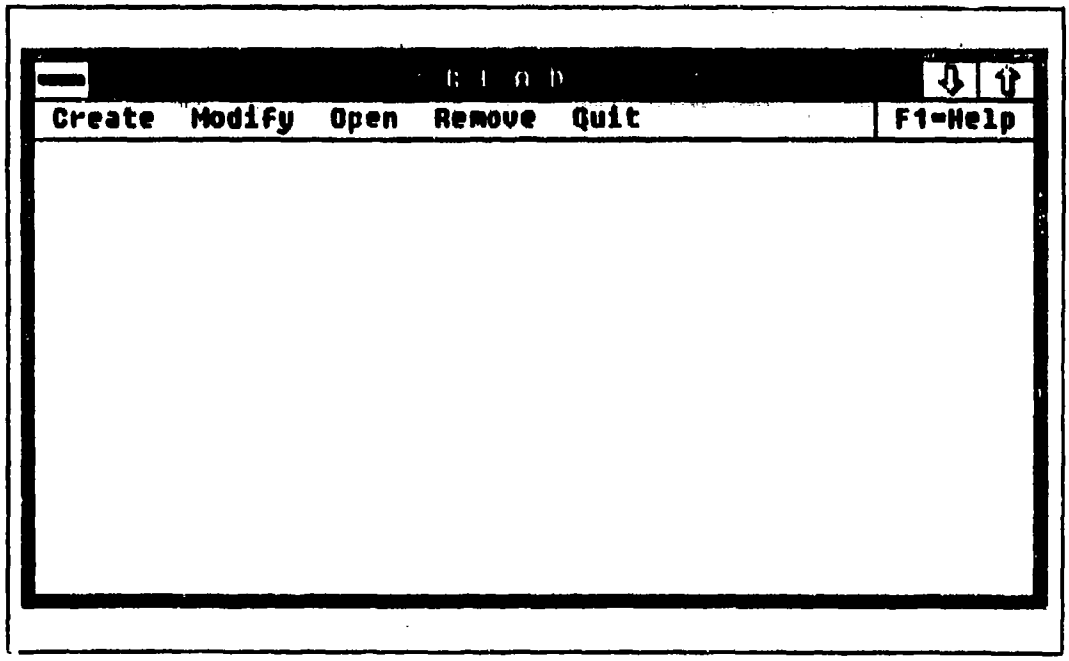



Figure 3.1 Standard Position of Help in GLAD Window

Another convention of the Window Programmers Reference Manual is the establishment of the **F1** key as the standard keyboard accelerator for access to help. A keyboard accelerator allows users to execute a command using the keyboard which can also be executed using the mouse. This convention of accessing help through the F1 key has been incorporated into GLAD. Design principle three states "be able to perform the same operation in more than one way". Thus, the GLAD user has been given two methods to access the help system. He can either select **F1=HELP** using the mouse from the Help window menu bar or use the keyboard accelerator **F1**. Regardless of the method used to access help, the **F1=Help** box flashes on the menu bar to give the user visual feedback that help has been accessed.

Using Windows provides screen options that make the help system more utilitarian. It is frustrating when the user is required to switch to a screen where his work is no longer visible in order to reference the on-line help, only to be required to switch back while remembering all of the pertinent information just viewed. Using an overlapping window allows the user to simultaneously view his work and refer to help in a separate window. The user will not have to memorize information before switching to his work screen. This ability to overlap and position a help window anywhere on the screen has been incorporated into the GLAD help system. The information provided must be neatly formatted and presented in such a manner that each screen explains a complete operation or concept. Forcing the user to scroll

through an unknown amount of information can cause "information anxiety" as in continuous scroll help systems.

Windows provides standard methods for closing windows which have been incorporated into the GLAD help system. The help window can be closed with the keystroke combination ALT_F4, clicking on the control-menu box and selecting close or by double-clicking on the control-menu box. These methods of closing the help window are consistent with all Windows-based programs. This consistency and flexibility adhere to design principles three and four, "perform the same operation in more than one way" and "be able to perform the logically equivalent operation in a consistent manner."

The problem then, is how to incorporate these design conventions into a help system for GLAD. Consideration was given to writing the entire help system using ACTOR, however, the amount of code needed could be unwieldy and require substantial amounts of time to generate. In addition, changes would demand that someone study and re-write the help system code for each change to GLAD. Using ACTOR would allow graphics to be incorporated into the help system; however, this could be difficult unless the graphics were limited to simple geometric figures such as circles and rectangles.

The decision was made to research the incorporation of GLAD with an existing program designed to produce a Windows-based help. This program would have to be capable of being easily integrated into GLAD Consideration was given to using

a stand alone help program called "HELP" produced by R Company (1988). This application would run independently of GLAD, but it would provide an easy method of providing textual help for Window-based applications. This was ruled out, because it did not allow the use of graphics within the help system. It was considered an inadequate solution for GLAD's help system.

## C. GUIDE AND GUIDANCE

A second application which was considered and ultimately selected for incorporation into the GLAD help system is OWL International's Guidance (1988). Guidance is a read-only Windows utility designed to allow users to display help while running a host application under Microsoft Windows. Guidance permits the integration of a help system which meets the design requirements of GLAD. Help information is contained in files referred to as *Guidelines*. These Guidelines can contain both text and bitmapped graphics. Guidelines are created using a program called Guide[*]. Guide is a general purpose hypertext document generator supplied with Guidance (Guidance Manual, 1988, p 9). Guidance not only furnished a simple means of incorporating text and graphics into the GLAD help system, but also provided the advantages of hypertext. ACTOR, the implementation language of GLAD, allows the easy integration of Guidance without excessive coding.

---

[*]Guide is a registered trademark of Owl International, Inc.

When accessed, Guidance opens an Index Guideline which is similar to a "Table of Contents". This Index then connects to either another part of the Index or to a Secondary Guideline, see Figure 3.2.

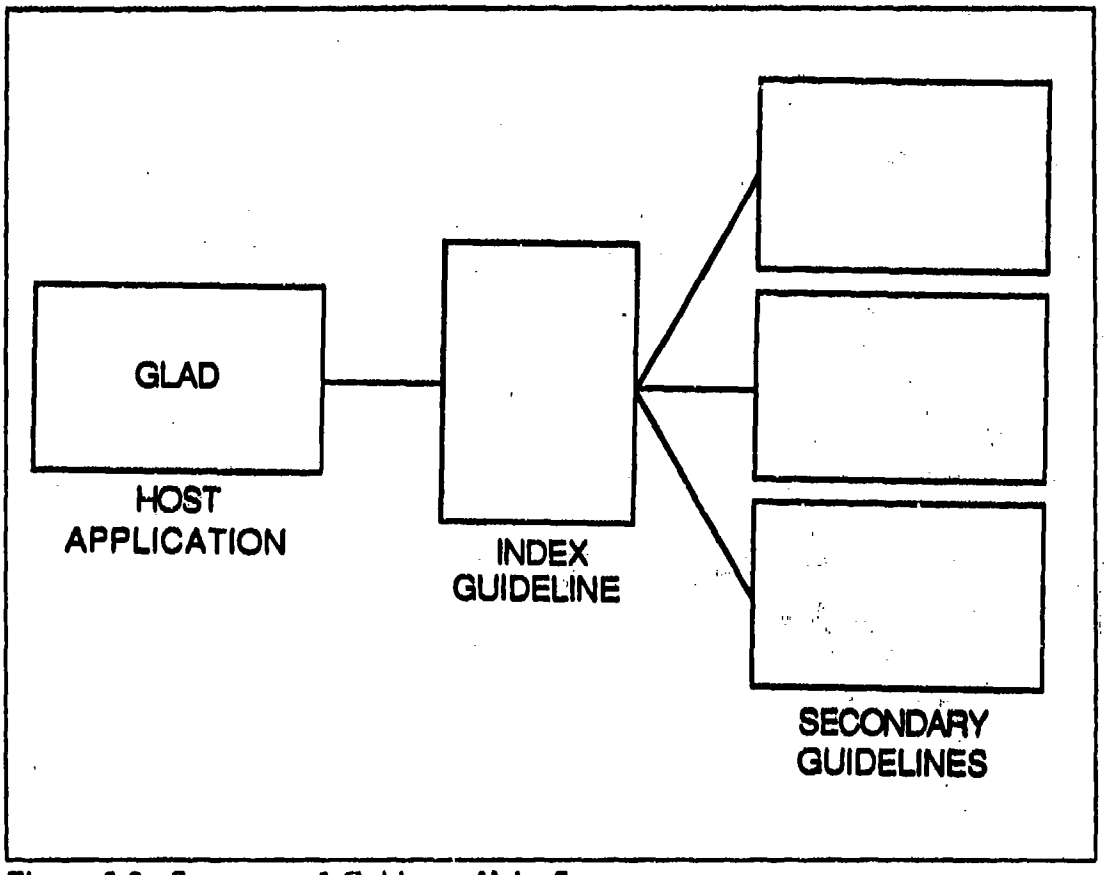



Figure 3.2 Structure of Guidance Help System

Guide furnishes four types of buttons as a means of accessing information within Guidelines: Reference, Expansion, Note and Command. These buttons are the building blocks which create the help system structure. Information contained within the help system which is linked to additional material is indicated by these buttons.

Reference buttons provide a means of linking material within a Guideline or between Guidelines. This powerful cross-referencing capability allows the user to move quickly to other areas of the help system. Information which is linked to a reference is indicated by the word or words of the Guideline shown in italics. When the cursor is positioned over a reference, the cursor changes to an arrow. See Figure 3.3 for the cursor shape of reference buttons. Graphics or areas of a graphic which are linked to a reference are also indicated by the cursor changing to an arrow. Clicking the left mouse button when the cursor is in this arrow shape will cause the reference link to be traversed and the additional information to be displayed.

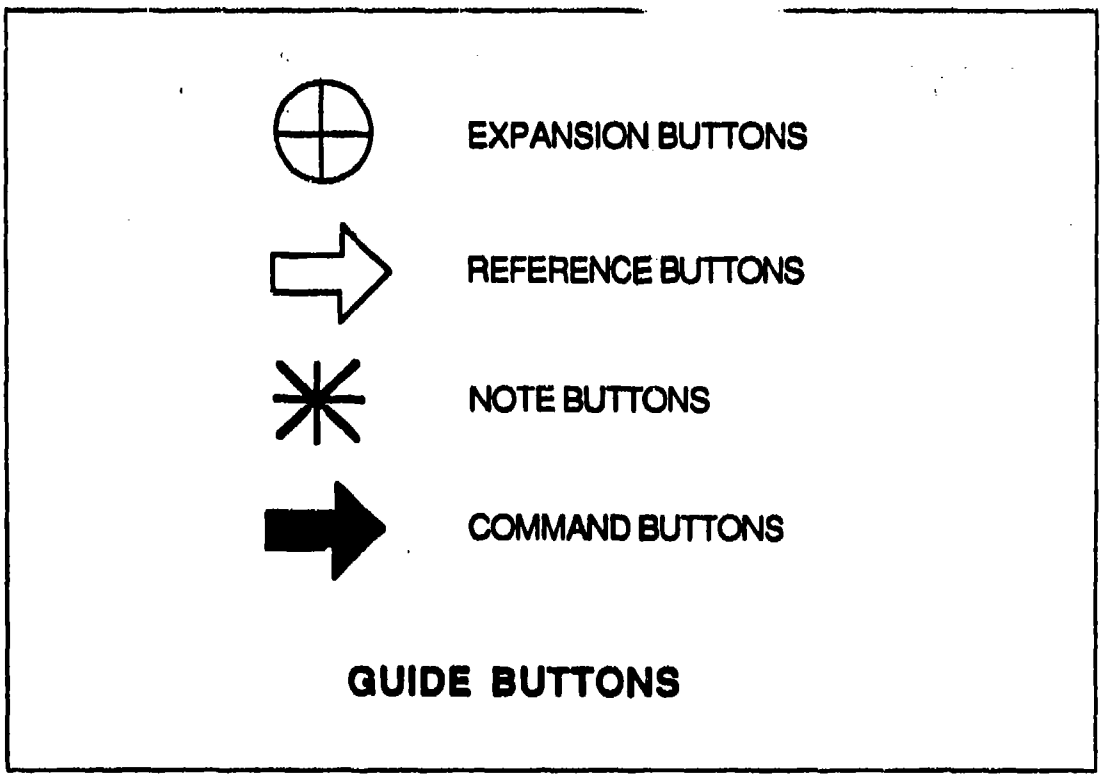



Figure 3.3 Shape of Guide Buttons

Expansion buttons allow information contained in the help system to be hierarchically nested. Text within a Guideline which is an expansion button is indicated in bold type. Positioning the cursor over this bold type will cause the cursor to change to a cross hair. See Figure 3.3 for the cursor shape of an Expansion button. Positioning the cursor over an area of a graphic which is an Expansion button also changes the cursor to a cross hair. When a user positions the cursor over an expansion button displaying the cross hair, clicking the left mouse button causes the information which is hidden below the button to appear. This information can be nested to an infinite number of levels. The user can go from level to level as desired, continually revealing more and more details.

Note buttons allow the user to access small pieces of supplementary information about a topic and are intended for temporary display. Text which is linked to a note is underlined. When the cursor is positioned over a Note button it changes to an asterisk. See Figure 3.3 for an example of the shape of the Note button cursor. When the cursor is positioned over a graphic which is linked to a Note button, the cursor changes to an asterisk. When the user clicks on a Note button a small overlapping window appears containing the additional information. This window remains visible only as long as the user continues to hold down the left mouse button. Note buttons are well suited to an on-line glossary, example formulas or short helpful hints.

Command buttons allow the user to launch other applications from within the help system. This would be useful if the user wanted to open up a text editor or a

spreadsheet to obtain information for entry or modification to a GLAD database. A user could also use this feature to open a text editor to jot down notes about the database he was using in GLAD. The cursor changes to a solid black arrow when positioned over a Command button. See Figure 3.3 for an example of the cursor shape of Command buttons.

Utilizing Guidance to implement a help system for GLAD has several other advantages not previously mentioned. Guidance allows the GLAD help system to be expanded or updated without requiring changes to the GLAD executable file. Guidance is designed to run in Microsoft Windows and is able to take full advantage of all of Windows' conventions. It furnishes multi-level capabilities through the use of hypertext. Guidance also provides functions which allow the help system to be context-sensitive. The next chapter will discuss the implementation details of incorporating Guidance into the GLAD help system.

# IV. GLAD HELP SYSTEM IMPLEMENTATION

## A. INTEGRATING GLAD AND GUIDANCE

Guidance takes advantage of a feature in Microsoft Windows which enables interapplication communication. This feature of Windows is referred to as Dynamic Data Exchange (DDE). Within Guidance is *Gydance.exe*, a dynamic linked library (DLL) or dynalink (Petzold, 1988, p. 805). Dynamic linked libraries are a feature of the Microsoft Windows environment that allow separate applications to dynamically share code. Each module is compiled and linked separately in an executable file. Utilizing the DLL, Gydance.exe, GLAD is able to communicate with Guidance through DDE. In essence, it allows Guidance to become an extension of the GLAD program. The advantages to this are that the application file is linked to help only at run time when needed, making the GLAD executable file smaller and less memory intensive. The link is also faster because only those modules which need to be linked are linked. Most importantly, several different applications can share the same resource, which is especially critical in a large integrated programming environment (Draganza, 1989, p. 59). ACTOR provides the library and procedure classes which allow easy integration of DLLs.

Cydance.exe furnishes three routines which allow GLAD to interface with Guidance. These routines are:

- GuidanceInitialise.
- GuidanceSetContext.
- GuidanceTerminate.

As the name implies, GuidanceInitialise initializes a link between GLAD and Guidance. Once established this link remains until the GuidanceTerminate routine is called. A link to Guidance must be established through GuidanceInitialise prior to any requests for context-sensitive help using the GuidanceSetContext routine. GuidanceInitialise must include the name of the index guideline. The GLAD index guideline is the file index.gui. All files associated with the help system are noted by the "*.gui*" extension.

Figure 4.1 shows the GLAD hierarchy. This figure indicates where the link to Guidance is established. The GuidanceInitialise routine is executed in the InitGuidance method of the GladWindow class. This method is executed each time a Glad Window object is created. Figure 4.2 shows the ACTOR code associated with the GuidanceInitialise routine. Notice that prior to executing the GuidanceInitialise routine, GLAD must create a new library and add the three routines provided by Guidance.

Each GLAD window which contains a menu bar provides access to help. When a user requests help by selecting "**F1=HELP**" from the menu bar, a help message is sent to the appropriate GLAD window. The help method of each window class includes a call to the GuidanceSetContext routine. This routine enables the help system to be

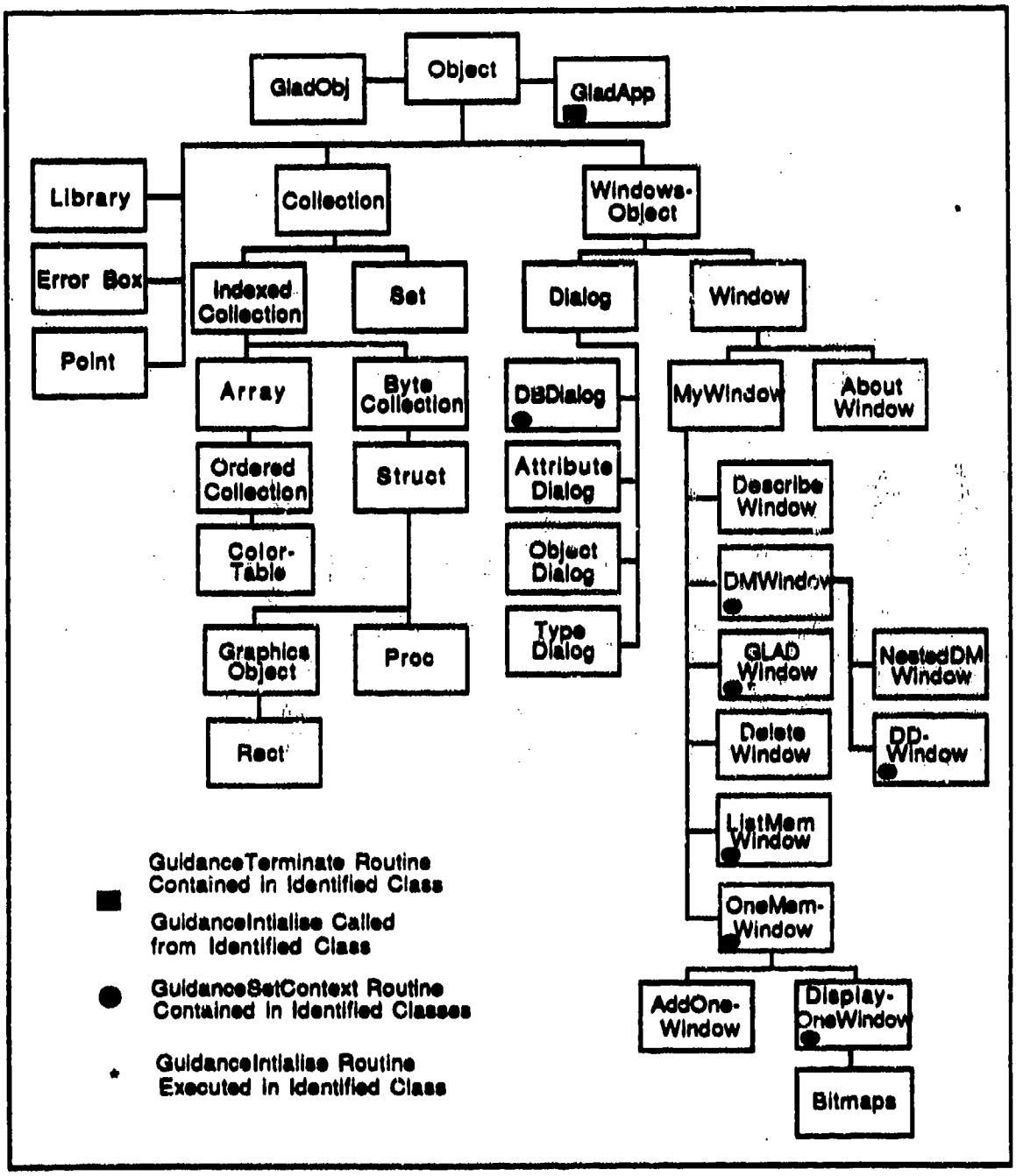



Figure 4.1 GLAD Hierarchy

```
/* Initialialse call to Guidance */
Def  initGuidance(selflaStr,aString)
{Lib := new(Library);
  Lib.name := "Gydance.exe";
  add(Lib, #GUIDANCEINITIALISE, 0, #(0 0 1 1 0 0));
  add(Lib, #GUIDANCESETCONTEXT, 0, #(0 1 0));
  add(Lib, #GUIDANCETERMINATE, 0, #(0));
  load(Lib);
  aString := "GLAD";
  aStr := "index.gui";
  HGuide :=pcall(Lib.procs[#GUIDANCEINITIALISE],
  HInstance, handle(self), lP(aString),
  lP(aStr), 1,1);
}!!
```

Figure 4.2 GuidanceInitialise Routine

context-sensitive. Included in this GuidanceSetContext call is a string that contains the name of the current GLAD window. Figure 4.3 contains some examples of the GLAD methods which utilize the GuidanceSetContext routine. Notice the name of the calling window is contained in a string in each method. Guidance searches the index guideline for this string, then traverses the link to the guideline containing information about the requesting window. This guideline then appears on the screen. See Figure 4.4.

Figure 4.1 also indicates the location of the GuidanceTerminate routine within the GLAD hierarchy. This routine is executed by the shouldClose method of the GladApp class. When the **shouldClose** message is sent to the Glad Application, the

```
GladWindow Class

Def topHelp(selflaStr)
{aStr :=asciiz("GLAD WINDOW");
  pcall(Lib.procs[#GUIDANCESETCONTEXT],HGuide,
  lP(aStr),1);
  freeHandle(aStr);
}

DMWindow Class

Def help(selflaStr)
{aStr :=asciiz("Data Manipulation Window");
  pcall(Lib.procs[#GUIDANCESETCONTEXT],HGuide,
  lP(aStr),1);
  freeHandle(aStr);
}

DDWindow Class

Def help(selflaStr)
{aStr :=asciiz("Data Definition Window");
  pcall(Lib.procs[#GUIDANCESETCONTEXT],HGuide,
  lP(aStr),1);
  freeHandle(aStr);
}

ListMemWindow Class

Def help(selflaStr)
{aStr :=asciiz("List Members Window");
  pcall(Lib.procs[#GUIDANCESETCONTEXT],HGuide,
  lP(aStr),1);
  freeHandle(aStr);
}
```

Figure 4.3 Example GLAD Methods Utilizing GuidanceSetContext

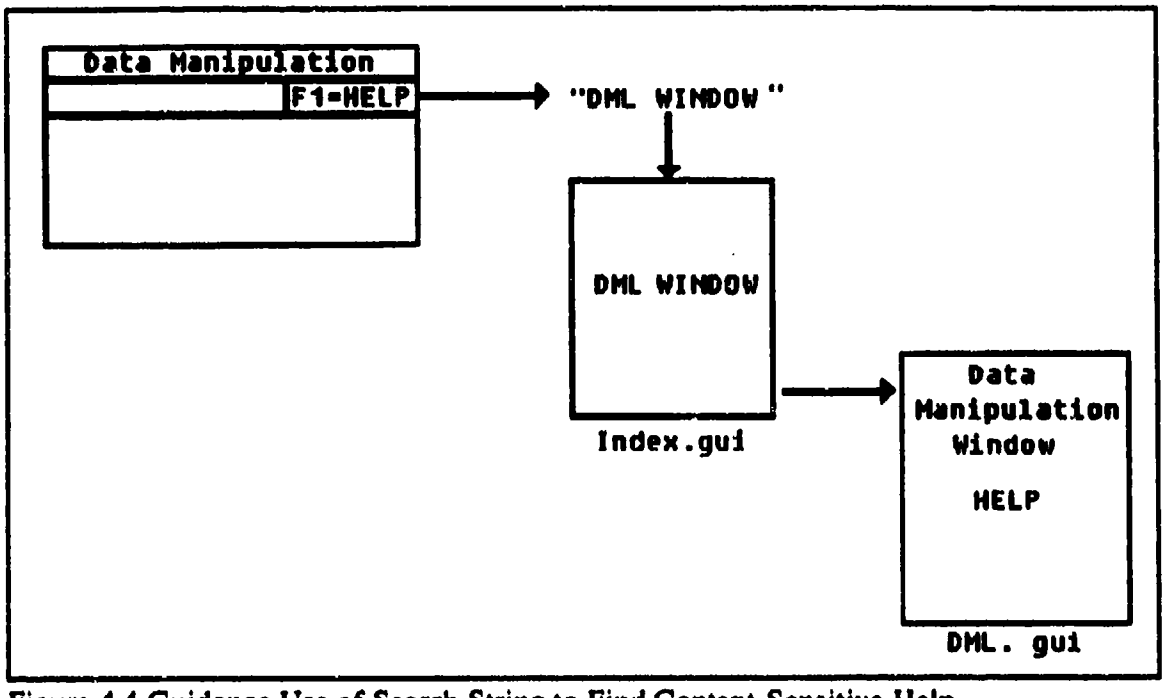


Figure 4.4 Guidance Use of Search String to Find Context-Sensitive Help

```
/* If any cleaning up needs to be done in the application before closing,
it should be done here. */
Def shouldClose(self)
{pcall(Lib.procs[#GUIDANCETERMINATE],HGuide)}!!
```

Figure 4.5 GladApp Class shouldClose Method

GuidanceTerminate routine is executed. Figure 4.5 depicts the shouldClose method GuidanceTerminate removes the link between Guidance and GLAD.

## B. IMPLEMENTING THE HELP SYSTEM ACCELERATOR KEY

Menus for GLAD windows are defined in the "resource script file". This is an ASCII file which contains GLAD's menus, dialogs, accelerator keys, icons and strings. (Duff, and others, 1989, p. 343). GLAD's resource script file is gladv02.rc. Incorporating the F1 accelerator key involves altering GLAD's resource script file. Appendix A contains a portion of gladv02.rc. Code to implement the F1 accelerator key is delineated in bold lettering. In the line

**\a"F1=Help", HELPER, HELP**

the (\a) causes "F1=Help" on the menu to be right justified. "HELPER" associates the identifier HELPER with this menu selection. The word HELP causes a box to be placed around the words "F1=Help". In gladv02.rc, the line

**VK_F1, HELPER, VIRTKEY**

associates the F1 key with the identifier HELPER. The line

**#define HELPER 950**

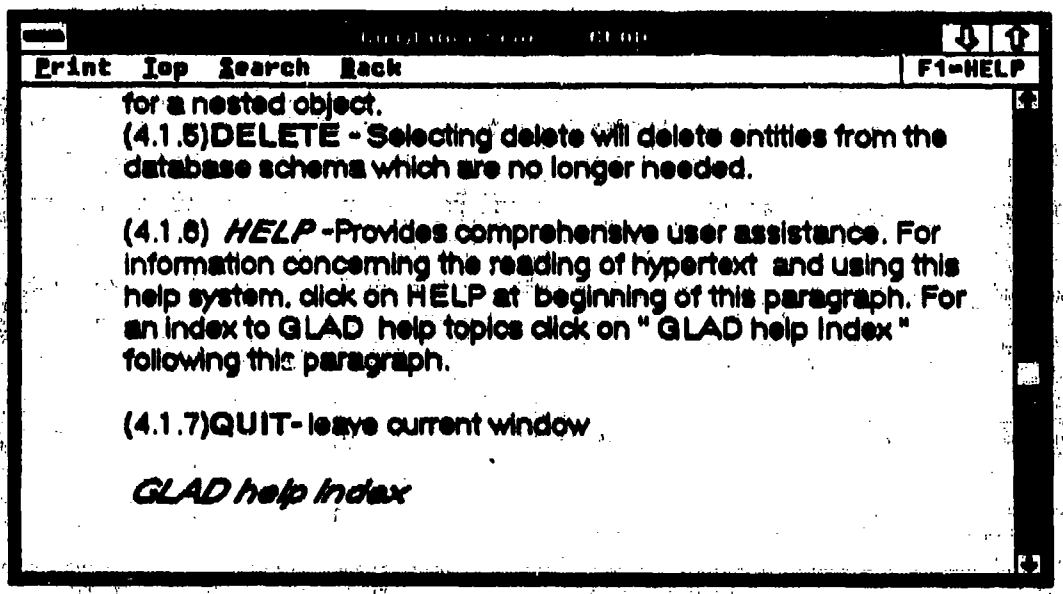
is contained in the file *glad.h*. The integer 950 can then be used as an index to the dictionary which contains the menu selections for each GLAD window. A dictionary in ACTOR is similar to an array in procedural programming languages. This number was arbitrarily chosen; however, it had to be higher than the number of menu options.

This is a standard Windows convention for calling help. For further explanation of RC and header files refer to Petzold (1988) . The InitMenuID method of each window class contains the dictionary which associates the identifier "HELPER" with the help method for each appropriate class. Appendix B contains the InitMenuID methods for each GLAD window class.

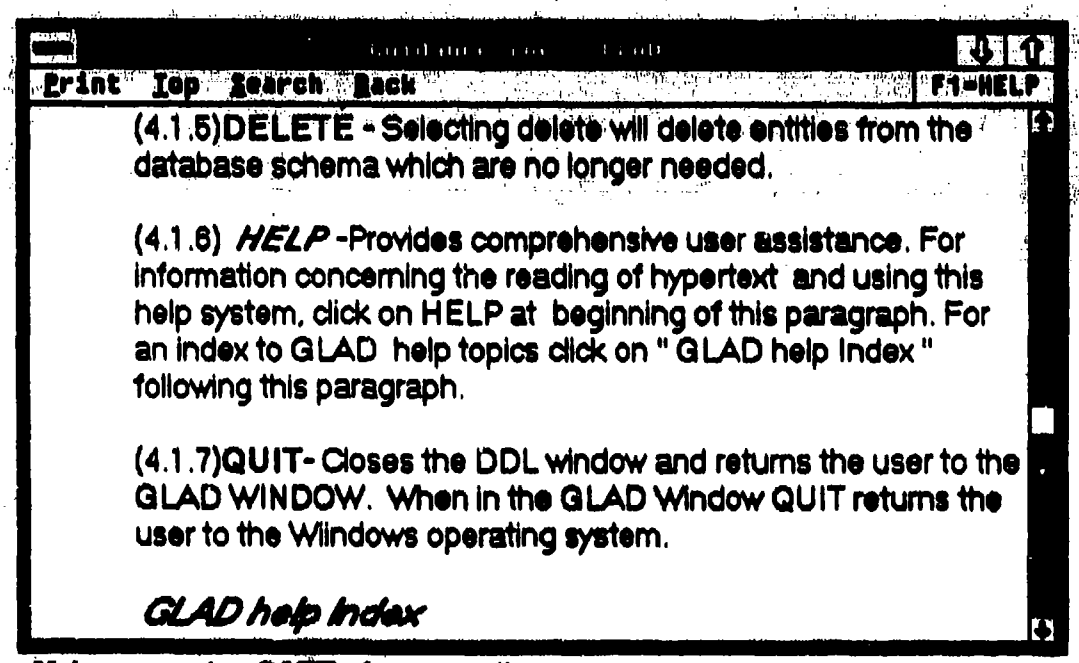
## C. GLAD HELP ORGANIZATION

The Guidelines for GLAD were designed to provide modules of information for each window within the GLAD hierarchy. These Guidelines were further developed to provide multi-leveled help within each Guideline. This enables the user to retrieve only as much information as desired. Figure 4.6 demonstrates how the user is able to access nested information through the use of expansion buttons. The top window of Figure 4.6 shows a section of the GLAD Data Definition Window Guideline. If the user desires additional information pertaining to the QUIT menu selection, he can obtain this information by positioning the cursor over the text "QUIT". The bold font visually indicates that the text "QUIT" is an expansion button. The cursor changing to a cross hair confirms that the text below is an expansion button. Clicking the left mouse button displays the information shown in the bottom window of Figure 4.6.

The modular design of the GLAD help system allows Guidelines to be altered with minimal effect on the remainder of the help system. Keeping the Guidelines small permits quick, easy reading and reduces the amount of memory required.




Help concernig QUIT prior to expanding




Help concerning QUIT after expanding

Figure 4.6 Demonstration of Expansion Buttons

The help system is designed to emulate the look and feel of GLAD. The Guideline for each GLAD window contains a replica of that window, see Figure 4.7. Ideally, the help system will behave identically to the GLAD program, except the user will be provided with help information when an operation is selected. This allows the user to visually associate a GLAD operation with a replica of that operation contained in the HELP Guidelines. For example, the user can obtain help by positioning the cursor over the item in the replica's menu bar that corresponds with the same menu selection in the GLAD window. Clicking the mouse on this item causes the help system to respond similarly to GLAD and provide help on that operation. Examples of GLAD operations will also be included whenever possible within help to make the information as clear as possible.

As mentioned in Chapter II, the potential to become disoriented while reading hypertext documents is a disadvantage of hypertext. A numbering system was incorporated into the GLAD help system in order to minimize user disorientation while accessing help. The numbering system allows the user to, at minimum, identify which Guideline he is using. The Guideline number corresponds to a number assigned in the Index Guideline. Figure 4.8 shows the GLAD index Guideline and its associated indices. Each successive nesting level within the Guideline will add a decimal point and a digit indicating the user's relative position in the Guideline. For example, 4.1.2 would indicate the user is two levels deep in the fourth Guideline. Figure 4.9 shows an example of the numbering system. Should the user become disoriented, Guidance

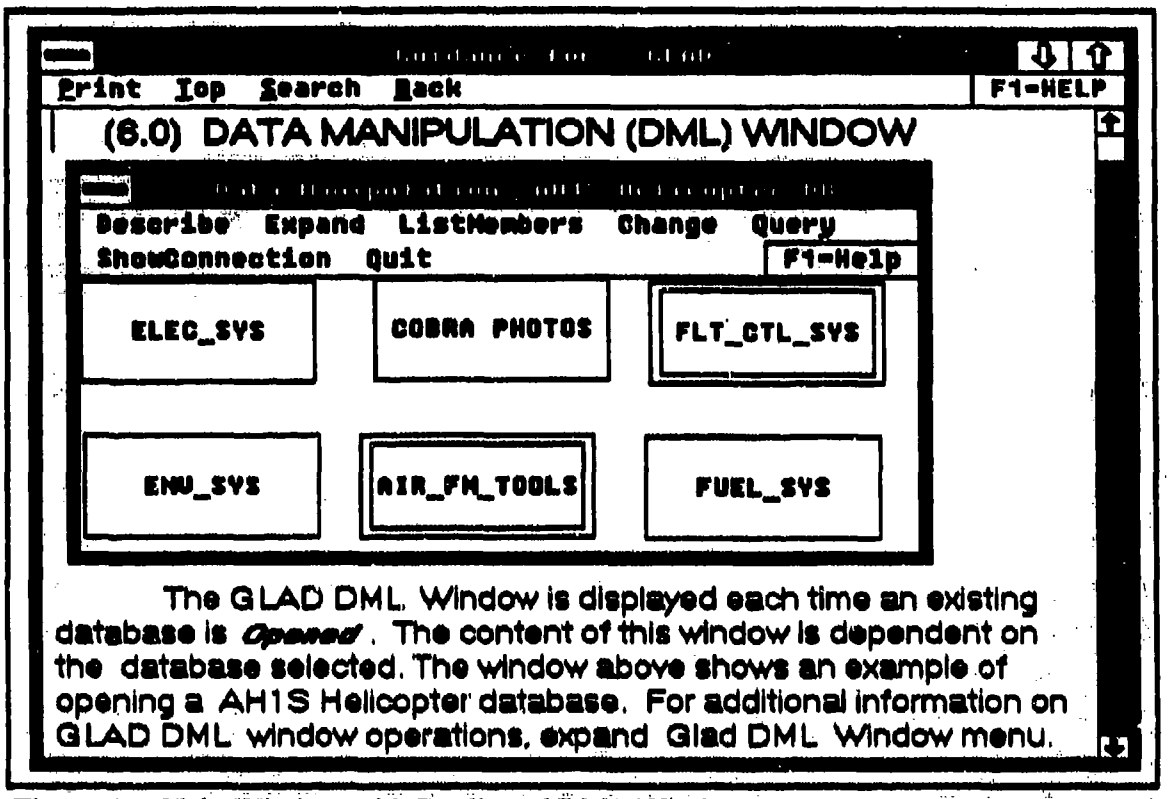



Figure 4.7 Help Window with Replica of DML Window

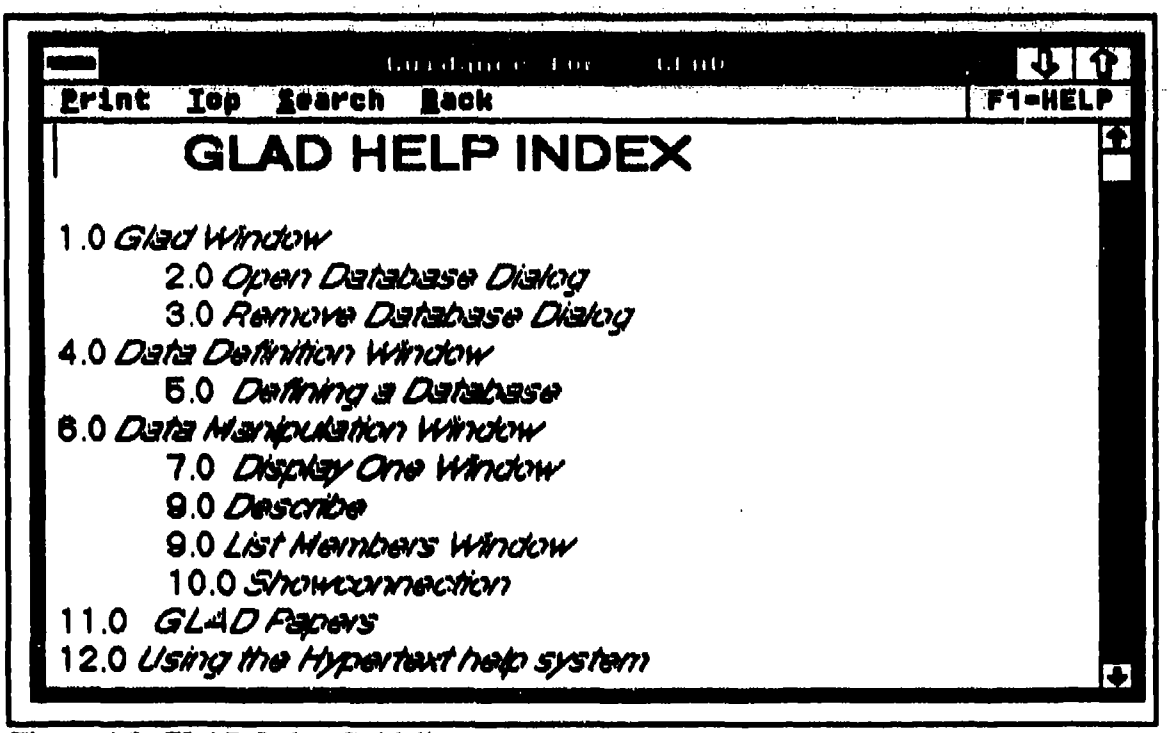



Figure 4.8 GLAD Index Guideline

provides the menu selection **BACK** on the help window menu bar. This allows the user to retrace the steps that he has taken within the Guideline. Guidance also provides a **TOP** menu selection. Selecting **TOP** automatically takes the user to the beginning of the Guideline. Links to the Index Guideline been have dispersed throughout the help system. This provides the user easy access to a position which is familiar should he become lost. In addition, it allows the user to access help in areas not contained in the current Guideline. This includes information concerning other GLAD Windows and operations, not contained in the current Guideline.

Should the user have a specific topic that requires explanation, he can use the search capabilities provided by Guidance. When the user selects **SEARCH** from any help window menu, a dialog box appears requesting the search topic. Guidance searches the current Guideline to locate information on this topic. If a string corresponding to the requested topic is contained in the current Guideline, the Guideline is displayed at that position of the information.

If the user desires a printed copy of the on-line help information, it can be printed by selecting **PRINT** from any GLAD help window menu bar. The document will be printed as it appears on screen. If an extended print-out of on-line information is desired, expansion buttons can be unfolded providing the full information available on the screen. If fewer details are required copy, only the desired information should be displayed, prior to selecting **PRINT.** Figure 4.10 displays an example of a GLAD help window with the **TOP, SEARCH,** and **PRINT** menu selections.

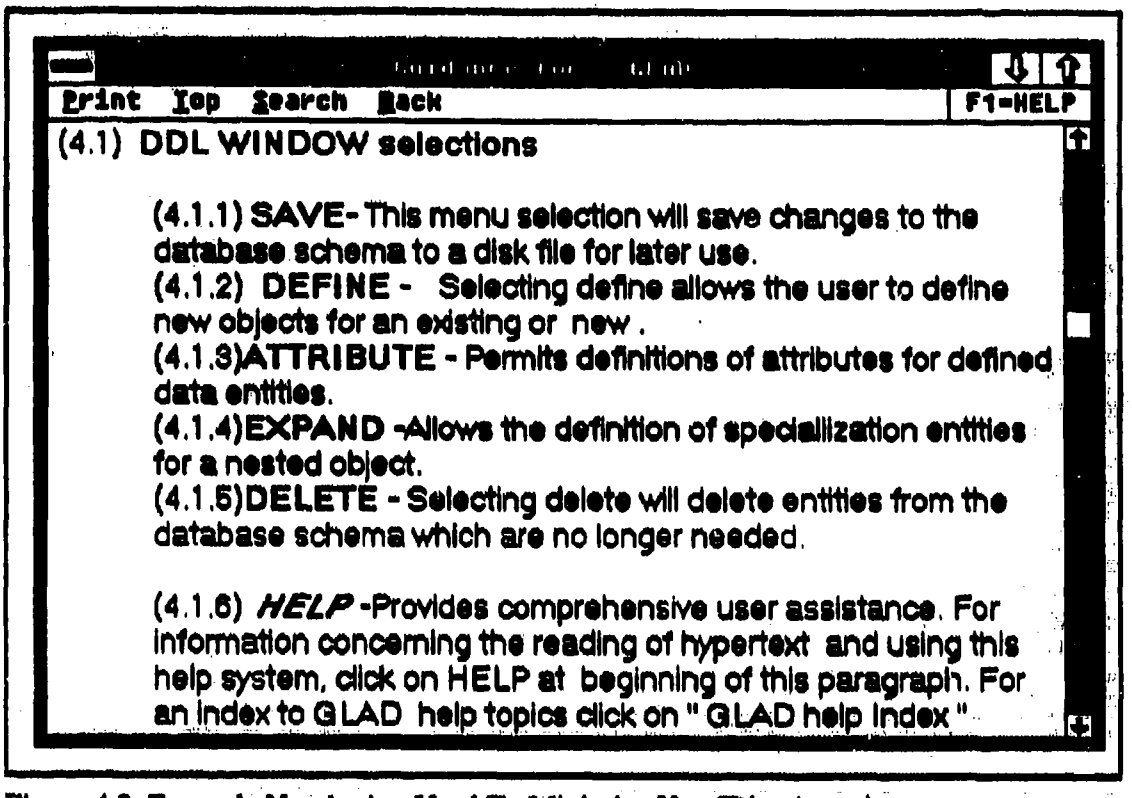



Figure 4.9 Example Numbering Used To Minimize User Disorientation

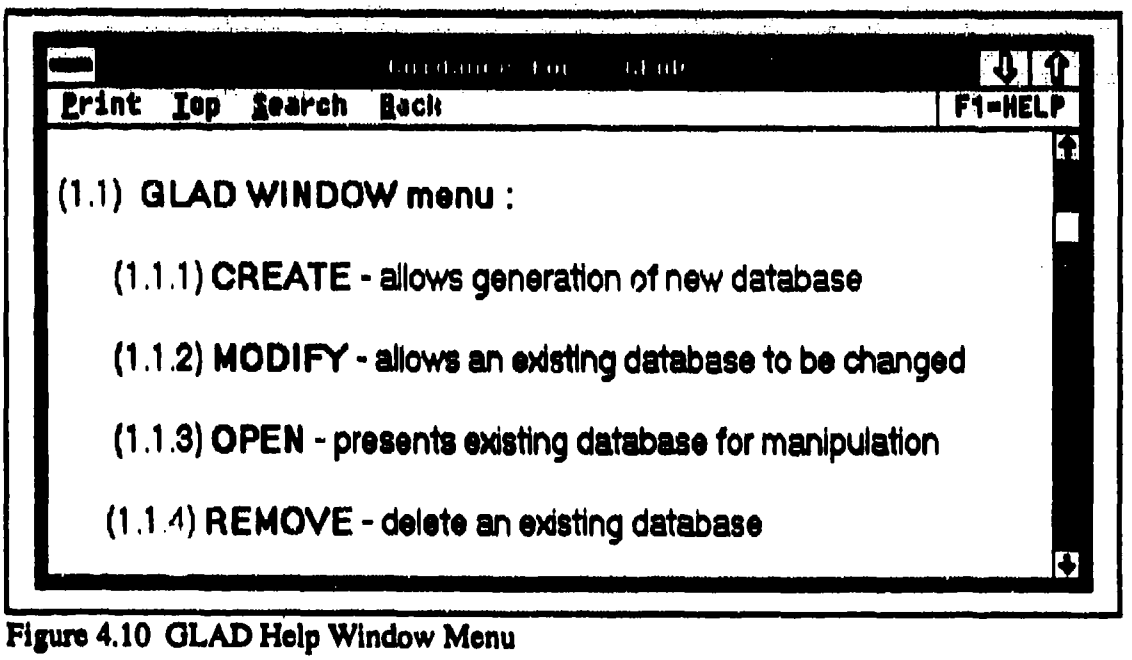



Figure 4.10 GLAD Help Window Menu

Any word or concept within the Guideline that may require further explanation is linked to a Note button. This serves as an on-line glossary of terms, providing the user quick explanations of unfamiliar terminology. Figure 4.11 example show an example of a Note button which is used to provide further explanation for the term "Select". Text which is a Note button is visually indicated by an underline.

Guidelines are constructed in accordance with the Guide and Guidance User Manuals. Help windows have been designed to present a complete description of an operation within a single window. This prevents the user from being required to scroll through multiple windows to obtain the information desired. This was not possible in all situations. Some operations required more than one screen to fully explain the operation.

## D. MEMORY MANAGEMENT

Throughout the development of the GLAD help system, memory management has been a troublesome issue. Attempts to integrate Guidance and GLAD within the ACTOR program environment on a 640K machine were unsuccessful. The only solution which allowed GLAD and Guidance to run simultaneously within the ACTOR programming environment was expanded memory. An additional megabyte of memory and the memory manager utility 386 Max' were required. This enabled GLAD and Guidance to run within the ACTOR programming environment. It was believed that

'386 Max is a trademark of Qualitas , Inc.

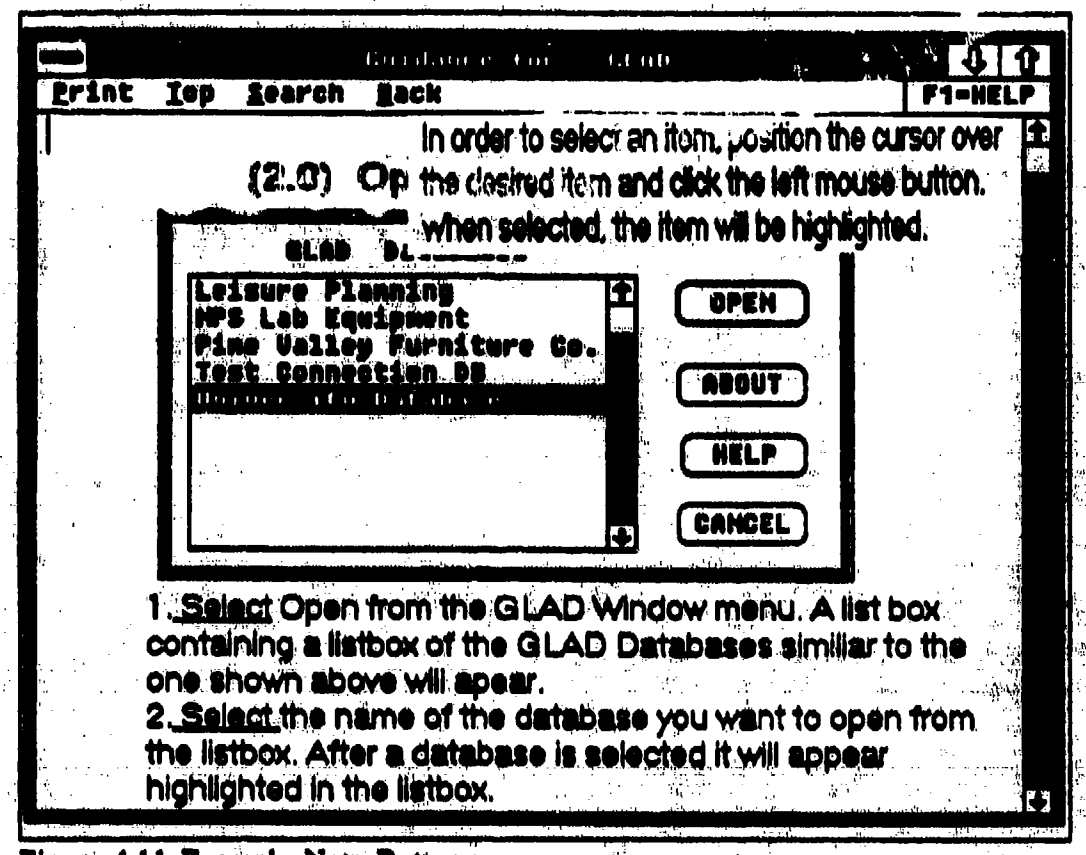



Figure 4.11 Example Note Button

once development was complete and a stand-alone version of GLAD produced, expanded memory would be unnecessary. Development of a stand-alone application using ACTOR involves a process of removing portions of ACTOR which are not used by the application. For example, the ACTOR Debugger, Editor and Browser were removed and consequently freed memory for GLAD. Once the code required to incorporate Guidance with GLAD was written and tested within the ACTOR environment, a stand-alone application was produced. Unfortunately, development of a stand-alone did not produce the desired results.

The help system operates correctly with one or two GLAD Windows open. Opening more than two windows reduces the amount of memory available to the help system to a point where memory is insufficient to correctly display the Guidelines. Insufficient memory results in an error message requesting the user to close one or more windows. If expanded memory is available, GLAD and the help system work perfectly together.

# V. CONCLUSIONS

## A. STRENGTHS AND WEAKNESSES

The purpose of this thesis was to design a help system for GLAD. The help system developed met the six design principles of GLAD while incorporating important features necessary in a help system. The strengths of the design are:

- The hypertext capability provides access to virtually endless amounts of information without reading unnecessary information.
- It is intuitive, easy to learn and use.
- It is powerful and cost effective in terms of time to implement.
- It is extensible.
- Minimal changes to GLAD were required to incorporate the help system.
- Graphics as well as text are easily incorporated.

The weaknesses of the proposed help system are:

- The help system will not run correctly without expanded memory.
- Despite the indexing system, the potential for the user to become disoriented within a hypertext document still exists.

The major weakness of this design is the requirement for expanded memory in order to achieve its full functionality. While the design constraints did not specifically state that this help system was limited to operation on a computer with 640K of memory, this is the most common memory capability of IBM compatible computers. The requirement for additional memory limits the use of this help system to computers that have expanded memory.

The implications of this requirement for additional memory may be that 640K is too limiting for a project the size and scope of GLAD. Intuitive, user friendly, graphic interfaces require resources, specifically memory. More memory is required as more features are added to a system. As GLAD expands, it will continue to require more memory. If 640K is an absolute requirement, it will not only limit the capabilities of the help system, but also the capabilities of the entire GLAD project.

The GLAD project must not be constrained by limiting memory to 640K. A choice must be made between an inferior help system which would allow GLAD to operate within 640K, or the proposed help system which is more capable, is easier to change and will better serve the needs of GLAD. The help system as proposed best suits the needs of the GLAD project. Restricting GLAD to a 640K of environment will result in a situation such that, as GLAD is developed and expanded, the help system will necessarily deteriorate.

The limitations associated with the 640K memory barrier imposed by the DOS operating system have been documented for at least five years. These limitations have become a driving force behind the development of operating systems such as OS2, which provide greater capabilities. This hurdle of memory limitation must be dealt with, in order to implement any help system as well as to develop GLAD to its fullest potential.

## B. FUTURE AREAS OF RESEARCH

Methods to make maximum use of available memory need to be explored. Possible areas of research include exploring development of GLAD with OS2 or UNIX to alleviate memory difficulties. Along with research into OS2 and UNIX, methods which would allow MS-DOS machines to take advantage of memory beyond 640K should be investigated. Investigations into optimizing the memory demanded by GLAD would also be beneficial.

Constructing hypertext documents to provide the best access to information for users is another area of possible research. Additional methods of indicating to the user where he is in the hypertext document need to be developed to eliminate the disorientation a user may experience when reading a hypertext document.

An intelligent help system that determines where the user is in a program and suggests courses of action or corrects mistakes is an area that deserves further exploration and research. Sound may also enhance the help system, as well as animation. These features require technology which is not currently available and may be too costly in terms of actual benefits to the GLAD project.

## APPENDIX A - SAMPLE SECTION OF GLADV02.RC

This appendix contains a portion of the file gladv02.rc. Only the section pertinent to the implementation of the help system menus is shown.

```
GladTopMenu  MENU
BEGIN
  MENUITEM  "Create", 1
  MENUITEM  "Modify", 2
  MENUITEM  "Open",   3
  MENUITEM  "Remove", 4
  MENUITEM  "Quit",   6
  MENUITEM  "\aF1=Help", HELPER, HELP
END


GladDmlMenu  MENU
BEGIN
  MENUITEM  "Describe",  1
  MENUITEM  "Expand",  2
  POPUP "ListMembers"
  BEGIN
 MENUITEM "All at Once",  3
 MENUITEM "One by One",  4
  END
  POPUP "Change"
  BEGIN
 MENUITEM "Add data",  5
 MENUITEM "Delete data",6
  MENUITEM "Modify data", 7
  END
  MENUITEM  "Query",       8
  MENUITEM  "ShowConnection", 9
  MENUITEM  "Quit",        11
  MENUITEM  "\aF1=Help", HELPER, HELP
END
```

```
GladDdlMenu  MENU
BEGIN
  MENUITEM  "Save",    1
  MENUITEM  "Define",  2
  MENUITEM  "Attribute", 3
  MENUITEM  "Expand",  4
  MENUITEM  "Delete",  5
  MENUITEM  "Quit",    7
  MENUITEM  "\aF1=Help", HELPER,HELP
END


GladLMMenu  MENU
BEGIN
  MENUITEM  "More",    1
  MENUITEM  "Modify", 2
  MENUITEM  "Quit",    4
  MENUITEM  "\aF1=Help",  HELPER,HELP
END

GladOMMenu  MENU
BEGIN
 MENUITEM  "Add",     1
 MENUITEM  "Delete", 2
 MENUITEM  "Modify", 3
 MENUITEM  "Prev",    4
 MENUITEM  "Next",  5
 POPUP  "GoTo"
 BEGIN
 MENUITEM   "First",  6
 MENUITEM   "Last",   7
  MENUITEM   "I th",   8
 END
 MENUITEM  "All",  9
 MENUITEM  "Quit", 11
 MENUITEM  "\aF1=Help", HELPER,HELP
 END
```

```
GLADV02 ACCELERATORS
BEGIN
  VK_INSERT, EDIT_PASTE, VIRTKEY
  VK_SUBTRACT, EDIT_CUT, VIRTKEY
  VK_ADD, EDIT_COPY, VIRTKEY

  VK_LEFT, VK_LEFT, VIRTKEY
  VK_UP, VK_UP, VIRTKEY
  VK_RIGHT, VK_RIGHT, VIRTKEY
  VK_DOWN, VK_DOWN, VIRTKEY

  "^a", EDIT_SELALL
  "^r", BR_REFORM
  "^z", BR_ZOOM

  VK_F1, HELPER, VIRTKEY
  VK_TAB, EDIT_TAB, VIRTKEY
  VK_PRIOR, EDIT_PRIOR, VIRTKEY
  VK_NEXT, EDIT_NEXT, VIRTKEY
  VK_HOME, EDIT_HOME, VIRTKEY
  VK_END, EDIT_END, VIRTKEY

  VK_DELETE, EDIT_CLEAR, VIRTKEY
  VK_DELETE, EDIT_CUT, VIRTKEY, SHIFT
  VK_INSERT, EDIT_COPY, VIRTKEY, CONTROL
  VK_INSERT, EDIT_PASTE, VIRTKEY, SHIFT
END
```

## APPENDIX B - GLAD INITMENUID METHODS

### GLAD WINDOW CLASS

```
Def initMenuID(self)
{
  menuID := %Dictionary ( 1->#makeNewDb
                          2->#modifyDb
                          3->#openDb
                          4->#removeDb
                          950->#topHelp
                          6->#close )
} !!
```

### DM WINDOW CLASS

```
Def initMenuID(self)
{
  menuID := %Dictionary( 1->#describe
                         2->#expand
                         3->#listMembers
                         4->#oneMember
                         5->#addMember
                         6->#deleteMember
                         7->#modifyMember
                         8->#query
                         9->#showConnection
                         950->#help
                         11->#close )
}!!
```

**DDWINDOW CLASS**

```
Def initMenuID(self)
{
  menuID := %Dictionary(1->#saveSchema,
                        2->#defineObj,
                        3->#attachAttr,
                        4->#defNestedObjects,
                        5->#deleteObj,
                        950->#help,
                        7->#quit)
}!!
```

**DISPLAY ONE WINDOW CLASS**

```
Def initMenuID(self)
{
  menuID := %Dictionary( 1->#addMember
                         2->#deleteMember
                         3->#modifyMember
                         4->#prev
                         5->#next
                         6->#first
                         7->#last
                         8->#goToIth
                         9->#allAtOnce
                         950->#help
                         11->#close)
}!!
```

**LIST MEMBERS WINDOW CLASS**

```
Def initMenuID(self)
{
  menuID := %Dictionary( 1->#more
                         2->#modify
                         950->#help
                         4->#close )
}!!
```

## LIST OF REFERENCES


Conklin, Jeff, "Hypertext: An Introduction and Survey", *IEEE*, September 1987.

Draganza, Michael, "Dynamic Link Libraries Under Windows", *Computer Language*, Vol. 6, no. 5, 1989.

Duff, Charles, and others, *Actor Language Manual*, The Whitewater Group, 1989.

Dumas, Joseph S., *Designing User Interfaces for Software*, Prentice Hall, 1988.

Galitz, Wilbert, *Handbook of Screen Format Design, 3rd Edition*, QED Information Sciences, Inc., 1989.

*Guidance: Hypertext Help System, Hypertext for Software Developers*, Owl International, Inc., 1988.

*Guide: Hypertext for the PC*, Owl International, Inc., 1988.

*Help*, R Company Ltd., 1988.

Jackson, Peter and Lefrere, Paul, "On the Application of Rule-based Techniques to the Design of Advice Giving Systems", *Int. Journal of Man-Machine Studies*, Vol. 20, 1983.

Kearsley, Greg, *Online Help Systems: Design and Implementation*, Ablex Publishing, 1988.

Killory, J.F., "Computer-Human Interaction and the Documentation Puzzles", *Computers and People*, Vol. 30, Nos. 5 & 6, 1987.

Meyer, Bertrand, *Object-Oriented Software Construction*, Prentice-Hall, 1988.

Petzold, Charles, *Programming Windows*, Microsoft Press, 1988.

Roberts, Roger, "Help - A Question Answering System", *AFIPS Conference* Proceedings, Vol. 37, Fall Joint Computer Conference, 1970.

Rowell, Michael, *The Suitability of an Object Oriented Language for Prototyping and Abstracting Data Types*, Master's Thesis, Naval Postgraduate School, June 1988.

Shneiderman, Ben, *Designing the User Interface, Strategies for Effective Human Computer Interaction*, Addison-Wesley Publishing, 1987.

Weiss, Edmond H., *How to Write a Usable User Manual*, ISI Press, 1985.

Williamson, Michael, *An Implementation of a Data Definition Facility for the Graphics Language for Database*, Master's Thesis, Naval Postgraduate School, December 1988.

Wu, C. Thomas, *GLAD: Graphics Language for Database*, Prepared for Chief of Naval Research, 1987.

Wu, C. Thomas and Hsiao, David K., *Implementation of Visual Database Interface Using an Object Oriented Language*, Presented at IFIP TC-2 Working Conference on Visual Database Systems, Tokyo, Japan, April 1989.

Wu, C. Thomas, "Benefits of Object-Oriented Programming to Implement a Visual Database Interface", *Case Studies of Object-Oriented Programming*, Addison-Wesley, Publication pending.

## INITIAL DISTRIBUTION LIST

| | | |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943-50022 | 2 |
| 3. | Commandant of the Marine Corps<br>Code TE 06<br>Headquarters, U.S. Marine Corps<br>Washington, D.C. 20360-0001 | 1 |
| 4. | Department Chairman, Code 52<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93943-5000 | 2 |
| 5. | Curriculum Officer, Code 37<br>Computer Technology<br>Naval Postgraduate School<br>Monterey, California 93943-5000 | 1 |
| 6. | Professor C. Thomas Wu, Code 52Hq<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, California 93943-5000 | 10 |
| 7. | Professor David Hsiao<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, California 93943-5000 | 1 |

8. Captain Lon M. Yeary
c/o Mr. Lon O. Yeary
2N236 Pearl Avenue
Glen Ellyn, Illinois 60137

2